Design Doc of BlackJack

1) Design of the classes

For each class, we listed 3 essential points of our design:

1. purpose of the class
2. benefit of this class to the existing design
3. benefit of this class, if any to a future game

Bet:

1. It records the bet of player in each round.
2. It clearly definite the bet of each player which is easy to calculate.
3. The bet can be further applied to other card games.

BlackJack:

1. It's the entrance of the whole program.
2. It helps set an entrance to start the game.
3. In the future, the class can be changed into other game version, which is easier to control.

BlackJackDealer:

1. To set a role in this game.
2. Separate dealer from the player which improve the structure of game.
3. The dealer can be easily changed to adjust other games.

BlackJackPlayer:

1. To set the role of player with their attributes and behaviors in this game.
2. player is an essential element in this game, it contains some variables and some operations the player might take.
3. The player can be easily changed to adjust other games.

BlackJackRules:

1. It defines the rules of this game.
2. It helps check whether the player wins or loses.
3. For future, the rules can be updated.

BackJackTable:

1. It defines where the participants play the Black Jack game.
2. Table is the controller of the game, it calls api from some instances and gathers needed information and then sends them to other api from some instances to let the game goes smoothly. Moreover, the table supports multi-players’ game by entering a number bigger than 1 when asked the number of player at the start of this game.
3. For future, some for features may be added to Black Jack table.

Card:

1. The class defines cards with suit, number and value.
2. It's the basic element of the card game which is essential to the whole program.
3. The cards can be further applied to other card games.

CardPlayer:

1. It defines all participants in the game, and is the super class of BlackJackPlayer and BlackJackDealer.
2. It defines some variables and methods that all card players have in common.
3. For future, we can add more features to card player and this class can be extends by other card game’s roles.

Config:

1. It contains some configuration of the system.
2. It provides some constant or static variables in the program.
3. For future, other static variables can be added to this class.

HandCard:

1. To define the cards that the player currently hold.
2. It maintains a List that allows player to split his handcard
3. For future, the handcard can support player to hold more than 2 hands of cards

Person:

1. To define a normal person with name.
2. It is the super class of CardPlayer.
3. For future, the more details such as birthday, nickname, some common methods and some new roles can be added.

Rules:

1. An interface of rules for the card game.
2. It defines one method ‘checkWin()’ for all specific rules related to some card games, this method is used to check the winner of the game.
3. For future, we can add much more required methods for all card games to this interface and it can be implemented by other games.

Shuffle:

1. It's the machine which controls all cards and gives card randomly.
2. With the Shuffle class, the player and dealer can get card randomly, the Shuffle class controls all the cards.
3. For future, the Shuffle can support more than 1 set of cards, just call Shuffle(int n) is ok.

Table:

1. An interface of table for the card game.
2. It defines two methods ‘playGame()’ and ‘printResult()’ for all specific table relating with some card games. The method playGame() is the logic controller of the game and the method printResult() prints the result of that game.
3. For future, we can add much more required methods for all card games to this interface.

Utils:

1. It contains some common methods in this program.
2. It includes all common operations of the system, so that it could be easily managed in the correct way.
3. For future, the utility methods can be added here.

Wallet:

1. To save or change the money of players
2. It defines some methods for the game to get or edit the money of players
3. For future, we can define different wallets for multiple players

2) Program running process

The program initials like this:

1. input how many player will play this game
2. input player's name
3. input player's money
4. choose whether the dealer is a real man or not
5. if the dealer is a real man, input the dealer's name

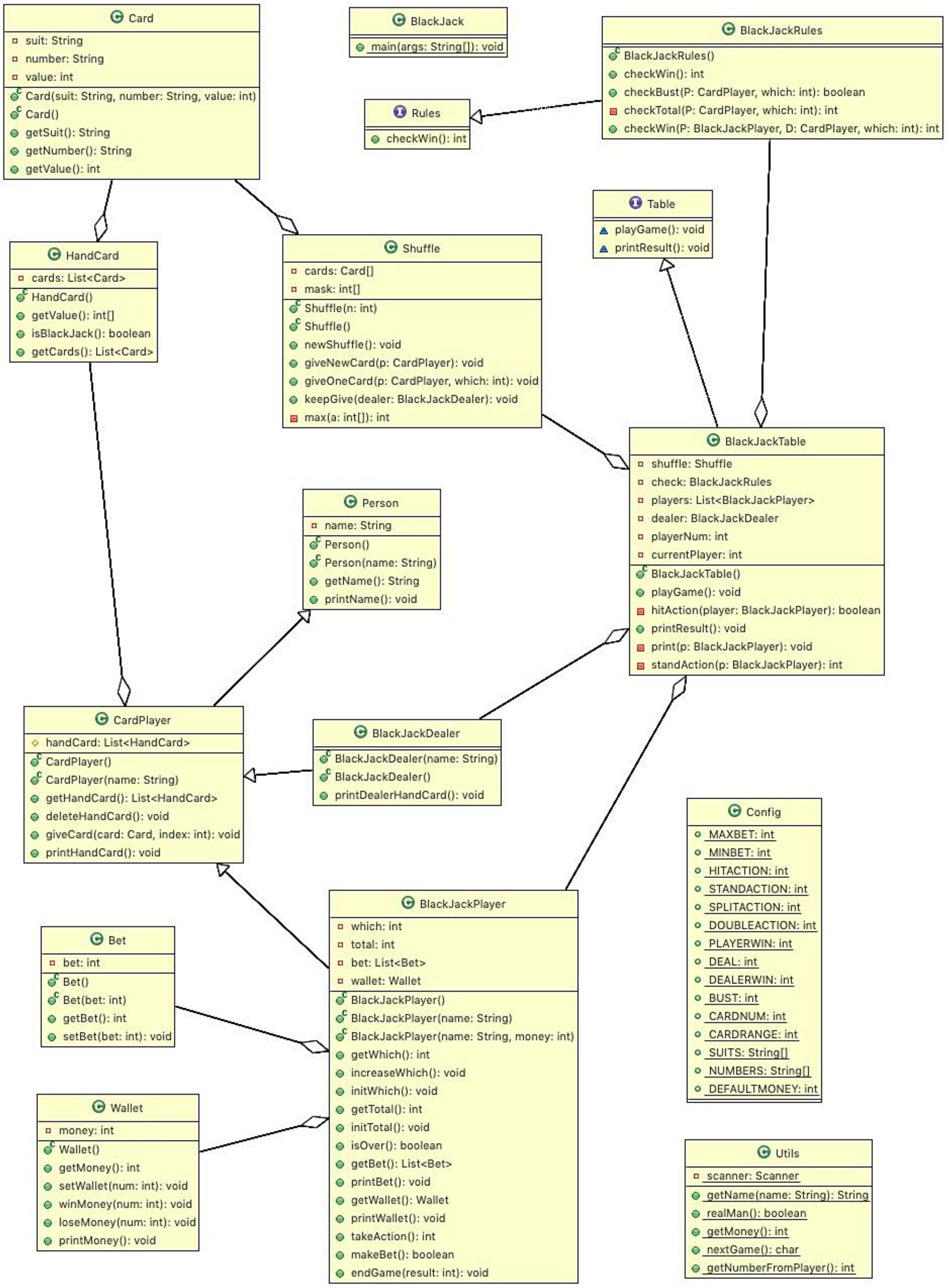
Each round of the Black Jack goes like this:

1. for each player, check whether he has more than $10 (the least bet is $10)
2. player inputs the money he wants to bet (more than $10 and less than $100)
3. player keeps choosing from four actions (hit, stand, split, double up)
4. the system gives us the result of this round, and finishes current round
5. system asks the player whether he wants to play a new game
    - if yes, go step 1
    - if no, remove the player from current game and print his final balance
6. if there is more than 1 player in the game, go back to step 1

At the end of the system: print the final money the player has

3) UML

Please refer to the next page.


Card
suit: String
number: String
value: int
Card(suit: String, number: String, value: int)
Card()
getSuit(): String
getNumber(): String
getValue(): int
BlackJack
main(args: String[]): void
BlackJackRules
BlackJackRules()
checkWin(): int
checkBust(P: CardPlayer, which: int): boolean
checkTotal(P: CardPlayer, which: int): int
checkWin(P: BlackJackPlayer, D: CardPlayer, which: int): int
Rules
checkWin(): int
Table
playGame(): void
printResult(): void
HandCard
cards: List<Card>
HandCard()
getValue(): int[]
isBlackJack(): boolean
getCards(): List<Card>
Shuffle
cards: Card[]
mask: int[]
Shuffle(n: int)
Shuffle()
newShuffle(): void
giveNewCard(p: CardPlayer): void
giveOneCard(p: CardPlayer, which: int): void
keepGive(dealer: BlackJackDealer): void
max(a: int[]): int
BlackJackTable
shuffle: Shuffle
check: BlackJackRules
players: List<BlackJackPlayer>
dealer: BlackJackDealer
playerNum: int
currentPlayer: int
BlackJackTable()
playGame(): void
hitAction(player: BlackJackPlayer): boolean
printResult(): void
print(p: BlackJackPlayer): void
standAction(p: BlackJackPlayer): int
Person
name: String
Person()
Person(name: String)
getName(): String
printName(): void
CardPlayer
handCard: List<HandCard>
CardPlayer()
CardPlayer(name: String)
getHandCard(): List<HandCard>
deleteHandCard(): void
giveCard(card: Card, index: int): void
printHandCard(): void
BlackJackDealer
BlackJackDealer(name: String)
BlackJackDealer()
printDealerHandCard(): void
Config
MAXBET: int
MINBET: int
HITACTION: int
STANDACTION: int
SPLITACTION: int
DOUBLEACTION: int
PLAYERWIN: int
DEAL: int
DEALERWIN: int
BUST: int
CARDNUM: int
CARDRANGE: int
SUITS: String[]
NUMBERS: String[]
DEFAULTMONEY: int
BlackJackPlayer
which: int
total: int
bet: List<Bet>
wallet: Wallet
BlackJackPlayer()
BlackJackPlayer(name: String)
BlackJackPlayer(name: String, money: int)
getWhich(): int
increaseWhich(): void
initWhich(): void
getTotal(): int
initTotal(): void
isOver(): boolean
getBet(): List<Bet>
printBet(): void
getWallet(): Wallet
printWallet(): void
takeAction(): int
makeBet(): boolean
endGame(result: int): void
Bet
bet: int
Bet()
Bet(bet: int)
getBet(): int
setBet(bet: int): void
Wallet
money: int
Wallet()
getMoney(): int
setWallet(num: int): void
winMoney(num: int): void
loseMoney(num: int): void
printMoney(): void
Utils
scanner: Scanner
getName(name: String): String
realMan(): boolean
getMoney(): int
nextGame(): char
getNumberFromPlayer(): int